A Note on Nested String Replacements

Holger Petersen
Reinsburgstr. 75
70197 Stuttgart
Germany

July 8, 2016

Abstract

We investigate the number of nested string replacements required to reduce a string of identical characters to one character.

1 Introduction

As part of a test data management project, every sequence of digits representing a number in certain strings stored in a data base had to be replaced by the single digit "1". This can easily be accomplished by replacing substrings matching $[0-9]+$ (representing any non-empty sequence of digits as an extended regular expression [1]) with 1. It however turned out that regular expression matching is very slow and that the syntax of functions making use of regular expressions varies between data base systems. Therefore it is good practice to manipulate strings using efficient and portable SQL-functions whenever possible.

The strings in question had a length of at most 32 characters. Therefore applying five nested `REPLACE`-functions each replacing 11 with 1 to an initial expression

$$\texttt{TRANSLATE}(s, \text{'023456789'}, \text{'111111111'})$$

would transform any sequence of digits in string s into 1.[1]

After some experiments, a solution using four nested `REPLACE`-functions was found by in turn replacing 1111, 111, 11, and again 11 with 1. Up to 34 digits can be reduced to a single 1 in this way and it is a natural question, whether a further improvement is possible.

[1] The function `TRANSLATE` substitutes single characters of its first argument, mapping each character appearing in its second argument to the corresponding character of the third argument. Characters in the second argument whithout a corresponding character in the third argument are removed (we will not use this feature). All remaining characters are not modified. The function `REPLACE` substitutes its third argument for all occurrences of the second in its first argument. Notice that `REPLACE` searches in a left-to-right manner and continues its search after a substited string. See [3] for further explanations and examples of `TRANSLATE` and `REPLACE`.

2 Results

Definition 1 *Task $R(m)$ is to replace all non-empty substrings 1^k for $1 \leq k \leq m$ in a string with 1 using nested* `REPLACE`*-functions with strings consisting of character 1.*

Lemma 1 *The inner-most* `REPLACE` *in a solution of $R(m)$ with minimum nesting for an $m \geq 2$ replaces 1^ℓ with 1^r for some $\ell \leq m$ and $r \geq 1$.*

Proof. If the replaced string of the inner-most `REPLACE` has at least $m + 1$ symbols, the `REPLACE` will not influence any string of length at most m and can be omitted from a solution of $R(m)$. If the substituted string is empty, at least one input of length at most m (namely 1^ℓ) is completely erased and $R(m)$ cannot be solved. □

Lemma 2 *The outer-most* `REPLACE` *in a solution of $R(m)$ with minimum nesting for $m \geq 2$ replaces 1^ℓ with 1^r for some $\ell \geq 2$ and $r \leq 1$.*

Proof. If the replaced string of the outer-most `REPLACE` consists of one symbol, then the substituted string cannot be empty or have a length greater than one. Therefore such a `REPLACE` leaves the text unchanged and would be redundant.

If the substituted string consists of more than one symbol, the `REPLACE` cannot be applied (since would not map to 1) and again woulkd be redundant. □

Proposition 1 *Task $R(3)$ cannot be solved with one* `REPLACE`*.*

Proof. By Lemmas 1 and 2 applied to the single `REPLACE` we only have to consider replacing 11 or 111 with 1. Either 111 or 11 would be mapped to 11, which shows the claim. □

Proposition 2 *Task $R(5)$ cannot be solved with two nested* `REPLACE`*-functions.*

Proof. Let the nested functions of a hypothetical solution of $R(5)$ be

$$\texttt{REPLACE}(\texttt{REPLACE}(s, \text{'}1^{\ell_1}\text{'}, \text{'}1^{r_1}\text{'}), \text{'}1^{\ell_2}\text{'}, \text{'}1^{r_2}\text{'}),$$

where s is the input. We will derive a contradiction for each possible choice of parameters ℓ_1, r_1, ℓ_2, and r_2.

By Lemma 1 we have $\ell_1 \leq 5$ and $r_1 \geq 1$ and by Lemma 2 we have $\ell_2 \geq 2$ and $r_2 \leq 1$.

If $\ell_1 \geq 4$, the strings 1, 11, and 111 are unchanged by the inner `REPLACE`. Then the outer `REPLACE` would have to map these strings to 1, which would be a solution of $R(3)$ with one `REPLACE` contradicting Proposition 1. We therefore only have to consider $1 \leq \ell_1 \leq 3$.

Let us assume $\ell_1 = 1$. If in addition $r_1 = 1$, the `REPLACE` would be redundant. Therefore $r_1 \geq 2$. The outer `REPLACE` maps 1^{r_1} directly to 1 in order to handle the input 1 or erases 1^d for a divisor $d \geq 2$ of $r_1 - 1$ leaving a remainder of one. In the former case 1^{2r_1} as the result of the inner `REPLACE` on input 11 would be mapped to 11. In the latter case 1^{2r_1} would be erased if $d = 2$ or mapped to 11 if $d \geq 3$. In each of these cases $R(5)$ is not solved.

If $\ell_1 = 2$, strings 1111 and 11111 are mapped to 1^{2r_1} and 1^{2r_1+1}. Both of these strings are then mapped to 1 by the outer `REPLACE`. If $r_2 = 0$ then $2r_1$ is divisible by $\ell_2 \geq 2$ in order to leave a single 1 from 1^{2r_1+1}. But then 1^{2r_1} is mapped to the empty string. If $r_2 = 1$ then $\ell_2 = 2r_1 + 1 \geq 3$, since otherwise a string with more than two symbols is generated from 1^{2r_1+1}. But now 1^{2r_1} with at least two symbols is not modified. In either case we derive a contradiction.

If finallly $\ell_1 = 3$, the input 11 is not changed by the inner `REPLACE` and $\ell_2 = 2, r_2 = 1$ in order to avoid the output 11. The input 11111 is mapped to $1^{r_1}11$ with at least three symbols by the inner `REPLACE` and to a string with at least two symbols by the outer `REPLACE` again contradicting the assumption. □

Notice that the bounds of Propositions 1 and 2 cannot be improved, since

$$\texttt{REPLACE}(s, \text{'11'}, \text{'1'})$$

and

$$\texttt{REPLACE}(\texttt{REPLACE}(s, \text{'11'}, \text{'1'}), \text{'11'}, \text{'1'})$$

solve $R(2)$ and $R(4)$ respectively.

Theorem 1 *With three nested* `REPLACE`*-functions $R(m)$ can be solved for any $m \geq 1$.*

Proof. Since $R(4)$ can be solved with two nested `REPLACE`-functions (and these could be extended by a redundant `REPLACE`), we only have to consider $m \geq 5$. The follwing sequence of replacements solves $R(m)$ for $m \geq 5$:

$$\texttt{REPLACE}(\texttt{REPLACE}(\texttt{REPLACE}(s, \text{'1'}, \text{'}1^{m-1}\text{'}), \text{'}1^{m}\text{'}, \text{'1'}), \text{'}1^{m-2}\text{'}, \text{''}).$$

The inner `REPLACE` blows up a block of $k \geq 1$ ones to length $k(m-1) = (k-1)m + (m-k)$. By replacing m symbols with 1 this is reduced to $(k-1)+(m-k) = m-1$ for $k \leq m$. Finally the outer `REPLACE` erases all but one symbol. □

3 Discussion

The somewhat surprising solution in the proof of Theorem 1 makes essential use of increasing the length of the input. If we allow length-decreasing replacements only, each `REPLACE` maps its input to strings covering a consecutive range of lengths and we can assume that the string being substituted is 1. By starting from the optimal solution of $R(4)$ with two nested `REPLACE`-functions (even if replacements are not not necessarily length-decreasing), an induction shows that $R(10)$ and $R(40)$ are the tasks that can be solved with three and four length-decreasing `REPLACE`-functions respectively. The sequence $2, 4, 10, 40$ appears as A159860 in the collection [2], where the recursive formula

$$a(n) = a(n-1)(a(n-1) + 6)/4$$

due to N. Sato is given for the maximum length $a(n)$ of a string of identical characters reducible to length one with n nested replacements (apparently length-decreasing in view of Theorem 1). The solution described in the Introduction is thus optimal with respect to nested `REPLACE`-functions under the additional assumption that all replacements are length-decreasing.

From a practical point of view the length-decreasing solution is apprroximately 40% faster than the one from Theorem 1, but the latter is still about twice as fast as a solution based on a regular expressions.

References

[1] Regular Expressions/POSIX-Extended Regular Expressions. https://en.wikibooks.org /wiki/Regular_Expressions/POSIX-Extended_Regular_Expressions (download July 6, 2016).

[2] The On-Line Encyclopedia of Integer Sequences®. https://oeis.org (download June 20, 2016).

[3] http://psoug.org/reference/translate_replace.html (download June 29, 2016).